The D0 framework and SAM on a Linux cluster.

Gabriele Garzoglio

July 2, 2001

Abstract:

SAM (Sequential Access through Meta-data) is the data access and job management system for the D0 high energy physics experiment at Fermilab. The D0 physicists perform data analysis by the use of a software framework integrated with SAM. SAM dispatches and controls user jobs through interfaces to underlying Batch Systems. In this white paper we discuss three issues of interest that enhance SAM functionalities, especially when running on a cluster of computers: the ability of running jobs in subsequent phases "piping" the output of a job as the input of the next; the integration of SAM with Root; the staging of the output files produced by the jobs on a cluster to a common location for debugging purposes.

Table of contents:

Introduction

SAM[I] offers already the possibility of running jobs on a cluster of computers. By talking to the users and the developers of SAM, I have found out that there are still a few issues of interest that are not addressed by the current implementation. The issues I will discuss in this white paper are all within the area of Knowledge and Distributed Intelligence[II] and focus on the use of SAM running jobs on a cluster of computers. Together with discussing the issues, I will propose possible designs to include these features in SAM.

I hope this white paper will be useful to set priorities on the issues discussed and lead to the implementation of some of the features described here.

As a note, a Linux cluster of 2 machines is available to test the developments of SAM discussed here. The cluster uses PBS[III] as batch system; yet, SAM does not implement an adapter to PBS. In order to use the cluster for development, the implementation of such an adapter should be done first.

Phases in the job processing

The current implementation of SAM allows the user to process only a single dataset at the time using either a d0framework[IV] analysis or a shell script. In case the output produced by such a job needs to be processed further, the human intervention is still needed to organize the input to the new job and to submit it. This procedure is not efficient, because the output of the first job may be available at a time when the user cannot submit the new job. It is also error prone, because the single user must do the organization of the new input each time "by hand". This leads to tempting but wrong methods of treating the files, such as using the files directly from the SAM cache without notifying SAM to lock them.

It is for these reasons that a general method of treating multiple phases in a job submission is desirable. A new command of the family "sam submit" can be implemented; it could accept a user job plan and an input dataset and treat the piping from one job to the next automatically.

In order to implement a robust system, we believe that the files produced at each intermediate step should be stored in SAM. As a supporting example, let us consider an analysis made of several steps that accepts in input several files and produces in output a single histogram. If the files are processed in parallel in a computer cluster and one of the nodes crashes, the information on the histogram will be partial. Yet, it may still be of interest and discarding the whole job may be a too drastic action. To be of any use in the general case, though, the user should be able to know global information about the dataset processed, like luminosity, in order to normalize the histogram. By looking at the composition of the intermediate datasets, the user could retrieve this kind information.

A drawback to this approach is that at every step a reliable network between the computer cluster, the SAM station, and the central database is needed to use the storage mechanism of SAM. In principle, since all the information for the various phases is processed within the cluster, such a reliable connection is not needed continuously for processing the whole chain. Glitches in the network could be responsible for holding jobs that otherwise would start immediately, without having to wait for the output of the previous jobs to be stored and then redelivered to them via SAM. In order to avoid issues like these, implementing a mechanism for transferring the intermediate information at a deferred time would be desirable.

Considering now the case where every file produced by the jobs is stored in SAM, in order to implement the mechanism of piping, the output files have to be grouped in datasets at each step. The choice of the dataset name is important to let the user keep track of what files have been produced at each phase of the process. First of all, a new data tier is needed to identify this kind of intermediate files. Then the dataset name should contain the list of processing script (name and version) up to that point. We believe that considering only the name of the script that produced the files is not enough to characterize the dataset. For example, considering an analysis where events are processed by different programs in subsequent phases, the final result may depend on the order in which the programs are applied. This is the case, for example, where the programs transform the variables non-linearly.

In the following section we describe the design of a possible implementation of the concept of job phases in SAM. We think of a scenario where the user submits a script to SAM; the script gets executed possibly on multiple nodes by the batch system that manages the Linux cluster; the Project Master is responsible for providing the input files to the jobs.

The user has to provide the list of scripts he/she wants to use and the initial dataset, via the modified SAM submit command. Also, each script has to store the output files in SAM. The new command will manage the piping, giving appropriate dataset names to the intermediate output files.

Two things must be added to the current implementation of SAM to enable the feature of processing phases:

1. the automatic definition of dataset from the output files at the end of each job
2. the scheduling of chained jobs to the batch system

In order to accomplish point 1, the user will have to store the files produced setting a dedicated meta-data dimension to the value of the project name that generated it. The project name is generated by the Station Master and it is unique. It is available to the user via the environment variable PROJECT_NAME.

When all the jobs are terminated and right before stopping the Project Master (see samscript.sh in the cvs package sam_user), a job-terminating script will be run. This script will define a dataset using the project name as the dimension to group the output files just produced.

In order to accomplish point 2 (see Figure 1) one can act at the time of submission to the batch system (see station_client.py in the cvs package sam_user). Currently, SAM submit is responsible to start the Project Master, ask the Station Master how to submit a job to the specific underlying batch system, submit the samscript.sh wrapper for the user script and check the status of submission via the Station Master.

The modified version could submit all the jobs at the same time, making sure that the batch system properly set their dependencies, i.e. each job must be hold until the job from the previous phase has finished[1]. Note that this is a new concept for the abstract interface to the batch systems and must be designed carefully.

Every job is submitted as a sam submit for a single script[2]. This has to be done to instantiate a new Project Master for every new dataset[3]. The intermediate dataset names can be defined here and stored in a temporary table to the database. The job-terminating script of point 1 can access this same table when defining the new datasets.

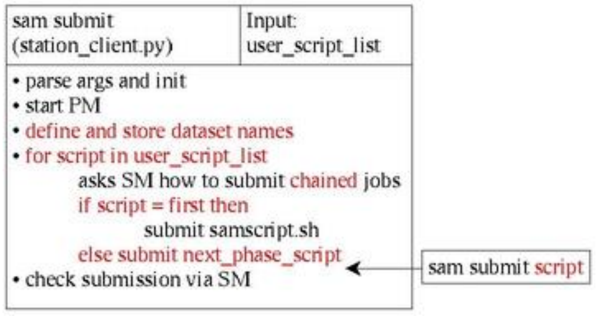
sam submit (station_client.py)
Input: user_script_list
• parse args and init
• start PM
• define and store dataset names
• for script in user_script_list
asks SM how to submit chained jobs
if script = first then
submit samscript.sh
else submit next_phase_script
• check submission via SM
sam submit script

Figure 1: The proposed way of submitting jobs to be executed in subsequent phases. The new SAM submit command submits them all at the same time and set job dependencies via the batch system. In red the parts that still need to be implemented; in black what is present already.

The integration of SAM and Root

Root[V] is going to be one of the platforms that the D0 experiment will use for data analysis. Also, the Root format is one of the two chosen to store the D0 thumbnails. Integrating SAM with Root is a project of interest by itself. Furthermore, considering the interest of the community in the development of the parallel version of Root, Proof, the SAM/Root integration could be the first step towards a general approach to data analysis in an environment where both data and jobs are fully distributed.

We propose three possible levels of integration. In the following paragraphs we

[1] All major batch systems provide this feature.

[2] Since the Project Master for the first job is instantiated already, the first job can be submitted directly to the batch system.

[3] Currently, there is no method to “reload” a list of files to an already instantiated Project Master.

will describe each of them, from most loose integration level but the fastest in terms of delivery, to the most complete but slowest (see paragraph “Schedule”). Note that in the latter, some issues about the maintenance of the software may raise, since Fermilab has no control over the development of Root.

Retrieving and locking in the SAM cache all the files needed for a Root analysis.

A popular way of using Root for analysis is making a Root-chain of the data files of interest. This technique allows a transparent random access to each event; generally, though, the access is performed by the use of an iterator, sequentially.

The most direct way of allowing users to chain Root files is having all of them present at the same time on disk. In our case, SAM could be used to retrieve the files to SAM caches before starting the Root analysis. The files should be then locked to prevent SAM from replacing them. At this point, the Root analysis could run on a chain of these files. When finished, the locks on the files should be released.

This strategy would be straightforward to implement in the case of a single user locking the files. Currently, there is no mechanism to manage concurrent locks in a multi-user environment. A crude, but easy way of implementing such a mechanism is using a counter instead of a flag to lock the files. This counter is the number of concurrent locks and is incremented/decremented by each process that needs the file. Only when the counter is 0 the file is available for replacement.

Note that some form of resource reservation should be implemented, as well. Without it, SAM could incur in dead locks: since all the files need to be present before starting the analysis, each new delivered file should be locked; therefore, two applications may never terminate waiting for the delivery of the files in a cache not large enough to keep them all and where all the files present are locked.

The development of a specialized Root chain class (TChain[VI]) for SAM in the D0framework.

The approach described above requires the allocation of a large amount of disk before starting any analysis. This is resource and time consuming. The approach proposed here allows the system to analyze the files while they are delivered to the cache, making this mechanism transparent to the user.

The D0framework is completely integrated with SAM; also, it should be possible to use Root classes from the framework, even if there is not much experience on how to do it, yet. We propose to develop a new framework class, TSAMChain, as a specialization of the Root class TChain[4]. This child class will overload the methods Add and LoadTree using the SAM commands that are already available and visible from the d0framework. Since the chain would be managed by TSAMChain, it would be possible to start a Root analysis on the files already present on cache, while transferring the missing ones, and to release the files that have been processed already, avoiding problems of dead locks.

Note that after gaining some experience, it should be already possible to run those analyses that do not rely on chains to process Root files from the d0framework. TSAMChain would make the file transfers more transparent and closer in style to Root.

The integration of SAM from within Root.

In the previous paragraph we have discussed the possibility of using Root from within the d0framework. All the SAM commands are available to the framework because of a C++ interface to SAM. In this paragraph we discuss the possibility of making available from within Root the SAM commands.

[4] This is the Root class that implements the methods to chain files together

In order to do this we propose to use the same C++ interface that is used by the d0framework. Some work is necessary to decouple the interface from the framework and to make a stand-alone shared library of it. Once this is done, we could generate a Root dictionary for the entry points of interest to the library: this would allow the use of SAM functionalities from the Root command line interface. Then we could start implementing functions that utilize SAM, like TSAMChain, from within Root. Note that if we plan to use the SAM interface from within functions only, we do not need to generate the Root dictionary.

Management of the output from the jobs for debugging purposes without using SAM

There are certain running conditions, like during debugging for examples, where the typical file size is small and where the user is interested in taking a look to the output without storing it. Typically, the user runs several cycles of debugging, producing many of these output files, the long-term utility of which is null. The ideal situation would be having all the files at a certain disk location at the end of the job, so that the user could analyze and discard them. We propose to modify SAM to enable the staging of these small size files to a common disk area, without storing them permanently.

Currently, when running on a cluster, the output files of each job are either stored on the local disk of the node where the job runs or stored via SAM. In our case, using SAM to store and retrieve the files is not an option because if the files are used once i.e. retrieved from storage, they cannot be deleted from the system anymore. Note that all the batch systems we consider to use allow the option of transferring output files when the file name is known at submission time. This is not in general the case, especially when running a d0framework analysis, where the I/O is managed at runtime by the framework using SAM.

In order to accomplish this task, we propose

1. to store the name and location of the files produced;
2. running a remote rcp to retrieve the files, at the end of the job.

In order to achieve 1 (see Figure 2), we propose to implement 2 new SAM commands, “sam remember name” and “sam remind names”. The user or the d0framework should call the first command after closing each output file. This command would simply store name and location of the files to a temporary table in the database using as a key the project name that produced it; the Project Master could do it in the user’s behalf.

The second command could be called by an output_script right before stopping the Project Master[5] (see samscript.sh in the cvs package sam_user). At this point the temporary table that holds the file name could be discarded.

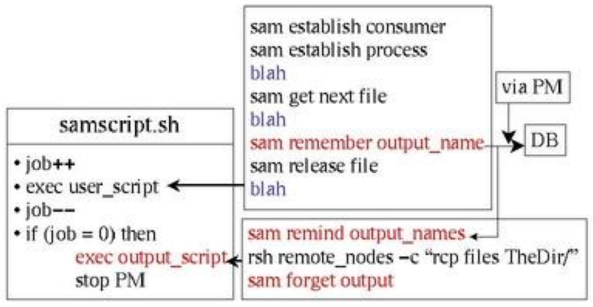

Figure 2: the proposed way of copying the output files produced by a series of jobs running on a cluster. samscript.sh is the wrapper for the user script submitted by SAM to the batch system. In red the commands to be implemented; in black what it is available already.

Schedule

The following is an estimate of the amount of time needed to implement each of the issues discussed above. Since not all of the

[5] The strategy is the same as for automatically defining a dataset of the output files (see paragraph “Phases in the job processing”), the script is different in this case.

details have been completely considered yet, this time is a tentative indication only. All of the following refer to implementation and testing.

- Setting up a SAM station running on a Linux cluster: a few days
- The adapter between SAM and PBS: 1 month
- Phases in the job processing: 1 month
- Retrieving and locking in the SAM cache all the files needed for a Root analysis: 2 weeks
- The development of a specialized Root chain class (TChain) for SAM in the D0framework: 5 weeks
- The integration of SAM from within Root: from 1 to 2 months
- Management of the output from the jobs for debugging purposes without using SAM: 3 weeks

Acknowledgement

This white paper is the result of many discussions: I want to thank Igor Terekhov and Sinisa Veseli for helping me speeding up my learning curve on sam and for the technical hints on the aspect of design described here; Philippe Canal for the help with Root; Lee Lueking for the time spent with me discussing software requirements; Vicky White and Ruth Pordes for the interest shown in this project.

References

[I] The SAM team, L.Lueking (co-leader), V.White (co-leader), L.Loebel-Carpenter, C.Moore, H.Schellman, I.Terekhov, J.Trumbo, S.Veseli, M.Vranicar. The home page `http://d0db.fnal.gov/sam`

[II] Knowledge and Distributed intelligence at Fermilab (KDI). The home page: `http://projetcs.fnal.gov/act/kdi.html`

[III] The Portable Batch System. The home page `http://www.openpbs.org`

[IV] *The D0 Framework*, talk given by J.Kowalkowski at The International Conference on Computing in High Energy and Nuclear Physics (CHEP2000), Sptember 2001

[V] Root, an Object-Oriented Data Analysis Framework; home page `http://root.cern.ch`

[VI] The Root Users Guide, v3.1b, Chapter 12 “Trees”, par “Chains”